

COMPUTER SCIENCE

C++ Arrays

In this tutorial, we will learn to work with arrays. We will learn to declare, initialize, and access array elements in C++ programming with the help of examples.

In C++, an array is a variable that can store multiple values of the same type. For example,

Suppose a class has 27 students, and we need to store the grades of all of them. Instead of creating 27 separate variables, we can simply create an array:

```
double grade[27];
```

Here, `grade` is an array that can hold a maximum of 27 elements of `double` type. In C++, the size and type of arrays cannot be changed after its declaration.

C++ Array Declaration

```
dataType arrayName[arraySize];
```

For example,

```
int x[6];
```

Here,

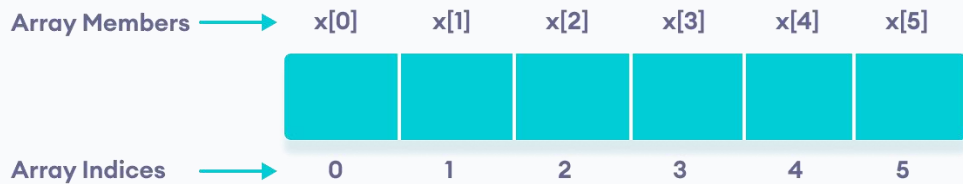
- `int` - type of element to be stored
- `x` - name of the array
- `6` - size of the array

Access Elements in C++ Array

In C++, each element in an array is associated with a number. The number is known as an array index. We can access elements of an array by using those indices.

```
// syntax to access array elements  
array[index];
```

Consider the array `x` we have seen above.



Elements of

an array in C++

Few Things to Remember:

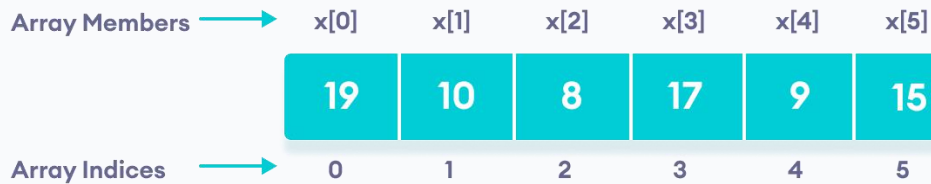
- The array indices start with 0. Meaning `x[0]` is the first element stored at index 0.
- If the size of an array is n , the last element is stored at index $(n-1)$. In this example, `x[5]` is the last element.
- Elements of an array have consecutive addresses. For example, suppose the starting address of `x[0]` is 2120d. Then, the address of the next element `x[1]` will be 2124d, the address of `x[2]` will be 2128d and so on.

Here, the size of each element is increased by 4. This is because the size of `int` is 4 bytes.

C++ Array Initialization

In C++, it's possible to initialize an array during declaration. For example,

```
// declare and initialize an array
int x[6] = {19, 10, 8, 17, 9, 15};
```



C++ Array

elements and their data

Another method to initialize array during declaration:

```
// declare and initialize an array
int x[] = {19, 10, 8, 17, 9, 15};
```

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

C++ Array With Empty Members

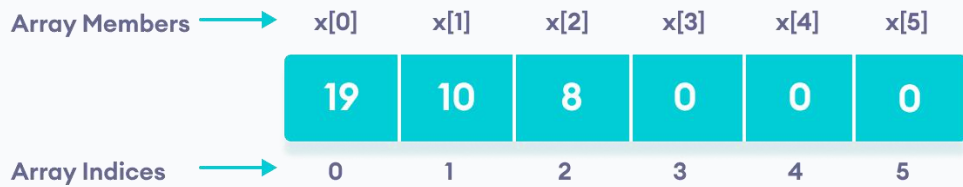
In C++, if an array has a size n , we can store upto n number of elements in the array. However, what will happen if we store less than n number of elements. For example,

```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

Here, the array x has a size of 6. However, we have initialized it with only 3 elements.

In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.

`x[6] = {19, 10, 8};`



Empty array

members are automatically assigned the value 0

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// change 4th element to 9
mark[3] = 9;

// take input from the user
// store the value at third position
cin >> mark[2];

// take input from the user
// insert at ith position
cin >> mark[i-1];

// print first element of the array
cout << mark[0];

// print ith element of the array
cout >> mark[i-1];
```

Example 1: Displaying Array Elements

```
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {7, 5, 6, 12, 35};

    cout << "The numbers are: ";

    // Printing array elements
    // using range based for loop
    for (const int &n : numbers) {
        cout << n << " ";
    }

    cout << "\nThe numbers are: ";

    // Printing array elements
    // using traditional for loop
    for (int i = 0; i < 5; ++i) {
        cout << numbers[i] << " ";
    }

    return 0;
}
```

Output

```
The numbers are: 7 5 6 12 35
The numbers are: 7 5 6 12 35
```

Here, we have used a `for` loop to iterate from `i = 0` to `i = 4`. In each iteration, we have printed `numbers[i]`.

We again used a range based `for` loop to print out the elements of the array. To learn more about this loop, check [C++ Ranged for Loop](#).

Note: In our range based loop, we have used the code `const int &n` instead of `int n` as the range declaration. However, the `const int &n` is more preferred because:

1. Using `int n` simply copies the array elements to the variable `n` during each iteration. This is not memory-efficient.

`&n`, however, uses the memory address of the array elements to access their data without copying them to a new variable. This is memory-efficient.

2. We are simply printing the array elements, not modifying them. Therefore, we use `const` so as not to accidentally change the values of the array.